AD-A255 376

Special Report
CMU/SEI-91-SR-13

②

Fault Tolerant Systems Practitioner's Workshop June 10-11, 1991

Walter L. Heimerdinger
Charles B. Weinstock

October 1991

DTIC
ELECTE
AUG 27 1992
S
A
D

416208

92-23734

Fault Tolerant Systems Practitioner's Workshop June 10-11, 1991

Walter L. Heimerdinger

Charles B. Weinstock

Systems Fault Tolerance

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
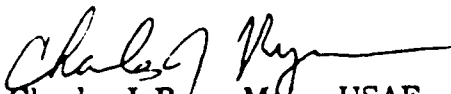
This document was prepared for the

SEI Joint Program Office
ESD/AVS
Hanscom AFB, MA 01731

The ideas and findings in this document should not be construed as an
official DoD position. It is published in the interest of scientific and
technical information exchange.

Review and Approval

This document has been reviewed and is approved for publication.

FOR THE COMMANDER

Charles J. Ryan, Major, USAF
SEI Joint Program Office

Table of Contents

Fault Tolerant Systems Practitioner's Workshop June 10-11, 1991

Abstract: On June 10-11, 1991, a Fault Tolerant Systems Practitioner's Workshop was held at the Software Engineering Institute. The purpose of the workshop was to attempt to identify how fault tolerance is being applied today, why fault tolerance is under used, and what can be done to bring fault tolerant practices into wider use. Attendance at the workshop was limited to a small number of practitioners who had successfully applied fault tolerance in a systems context. This report summarizes the proceedings of the workshop which included a discussion of barriers to the deployment of fault tolerant systems, a summary of the state of the practice, and a discussion of the technology needs of fault tolerance. The report concludes with a discussion of ways the Software Engineering Institute may be able to help bring fault tolerant practices into wider use.

1 Introduction

1.1 Workshop Objectives

A Fault Tolerant Systems Practitioners' Workshop was convened at the SEI on June 10-11, 1991 to obtain a sample of the state of the practice in fault tolerant system design and implementation. The purpose of the workshop was to attempt to identify how fault tolerance is being applied today, why fault tolerance is underused, and what can be done to bring fault tolerant practices into wider use.

The workshop was convened by the SEI Systems Fault Tolerance Project. The project is attempting to characterize both the state of the art in fault tolerance technology (the concepts and techniques generally recognized as part of the technology) and the state of the practice (the concepts and measures actually used in the development of deployed fault tolerant systems). While the state of the art is generally discussed in the literature, proprietary and business pressures, and the lack of incentive to publish make the state of the practice more difficult to assess; hence this workshop.

1.2 Workshop Format

Attendance at the workshop was limited to a small number of practitioners who had successfully applied fault tolerance in a systems context. The majority of invited participants were from industry since the activities of academic researchers are more widely available. Each attendee was experienced in the implementation of a deployed fault tolerant system using both hardware and software techniques. Invited attendees were limited to practitio-

ners in real-time fault tolerant applications, rather than in applications that emphasized database technology, which is also an important area for future consideration.

To stimulate discussion, the SEI circulated a set of barriers which appear to limit the wide-scale adoption of fault tolerant techniques. A goal of the workshop was to identify additional barriers to progress as well as possible ways of overcoming these barriers. To this end, each participant was asked to submit a one- to two-page position paper on one or more of the following issues:

- the barriers mentioned in the announcement,

- the application of fault tolerance to a system development,

- the problems of using fault tolerance in development contracts, and

- possible solutions to the above problems.

1.3 Chronology

The workshop began with an overview of the SEI, and the Systems Fault Tolerance Project. Then, each participant was given the opportunity to present his views, with ample time for discussion. The participants then worked together as a group to summarize the state of the practice in applying fault tolerance, major problems, and potential solutions, in list form. This list was used to create this summary report.

2　Barriers to the Deployment of Fault Tolerant Systems

In the workshop announcement, the SEI suggested that there were several barriers to progress in the deployment of fault tolerant systems. Participants were asked to comment on them. The following section lists the barriers suggested by the SEI, along with explanatory comments. This is followed by comments and additional barriers from the workshop participants.

2.1　Barriers Proposed by the SEI

Fault tolerance is not a "top-of-the-list" consideration.

Managers don't understand the importance of fault tolerance in achieving dependable systems. Dependability must become as important a consideration as performance, quality, and cost. The need for fault tolerance and maturity of the technology must become widely accepted so that it is risky *not* to incorporate fault tolerance.

No convenient metrics of the dependability of a system exist.

If dependability can't be conveniently measured, it is difficult to tell whether it has been achieved, or whether one system is more dependable than another. Contracts are made specifying "seven nines" reliability (i.e. probability of failure of 10^{-9}) with no way of telling whether the dependability goal has been met. Contractors need clear, *measurable* goals.

Program managers don't have the tools to properly specify fault tolerance in their contracts.

This is related to the previous two barriers. No program manager can be an expert in every area; yet, if a program manager lacks knowledge in this area, little help is available. As a result contractor and approach are often selected before dependability requirements have been properly specified.

There is an "all or nothing" mentality when specifying fault tolerance.

"All or nothing" means that if any component of the system fails, the entire system is said to fail. To achieve this level of fault tolerance is extremely expensive, if not impossible. In reality there are very few systems that couldn't be partitioned into functions that are highly critical (and must therefore always work), and those that are less critical (and could be jettisoned in the event of failure).

Fault tolerance is included in a system design for two similar but very different reasons: safety and dependability. In the former case extreme care and substantial cost is usually justified. In the latter, affordability becomes more important.

There is an "all or nothing" mentality with regard to which faults will be tolerated.

It is much too costly to build a system that attempts to cope with all faults whether or not they are likely to occur. Instead it is important to specify and design a system which copes with specific faults or hazards based on an understanding of the application. Using a priori knowledge of the environment and possible failures (or *credible faults*) can reduce the cost and actually result in a system which one feels better about. It is important to spend time at the beginning of a project assigning priorities to fault tolerance needs. This leads to:

There is insufficient failure data available to designers of fault-tolerant systems.

Few organizations have collected failure data. Since the data provides them with a competitive advantage, those that have are reluctant to share.

The lack of examples of software fault tolerance inhibits the development of fault tolerant software systems.

There are many safety systems that are implemented as analog systems but that cry out for the use of computers (e.g. those for nuclear reactors). They aren't computerized because we don't trust them as well.

Each major fault tolerant system design begins anew.

Fault tolerance techniques supplement fault avoidance and fault removal in achieving dependable systems. Fault avoidance relies on the use of tried and true designs and repeated use of designs contributes to fault removal. Although many of the basic principles are reused in the designs of new fault tolerant systems, each individual system usually is a completely new design. Thus, fault tolerance sacrifices the advantages of fault avoidance and fault removal.

2.2 Additional Barriers Identified at the Workshop

The workshop participants identified a number of barriers in addition to those listed above. These include:

A need to educate users, especially government program personnel

Potential users of fault tolerance need to understand that it is a means for achieving dependability in their systems and not an end in itself. They also need to learn when fault tolerance may not be appropriate. They need to recognize the importance of incorporating fault tolerance early in the design process. Also, they need to understand the relationship between fault tolerance and reliability and how to specify fault coverage and system reliability parameters.

Dependability/fault tolerance are not included in educational curricula.

Although operating systems technology, compiler technology and artificial intelligence are covered in many degree programs, with few exceptions there are no courses offered in de-

pendability and fault tolerance in university-level programs. As a result, new graduates are ill prepared to deal with the complexities of such systems, and indeed are not sensitive to the need to design systems with dependability in mind. The University of Cincinnati's major in Dependable Systems Engineering is an important exception.

A better understanding of fault tolerance economics is necessary to make intelligent decisions in dependable systems design.

This is reflected in the tendency of some programs to rely entirely on fault prevention (i.e. conservative software design and an extensive quality process) to develop critical software. Also, it is necessary to understand the relationship of fault tolerant software with other fault tolerance measures, including manual backup mechanisms.

Intuition is a poor guide when dealing with the extremely rare events characteristic of ultra-dependable systems.

Failures that persist in ultra-dependable systems are not simple: they usually involve complex interactions and multiple failures, and are extremely difficult to understand. Consequently, simple analysis based on straightforward scenarios involving single failures cannot guarantee successful recovery in such systems.

The accounting community does not have the tools to measure the value of fault tolerance.

The accounting community strongly influences program decisions. This community is unable or unwilling to measure the value of fault tolerance: hence fault tolerance appears as an unnecessary cost. This is at least partially due to the difficulty of estimating the value and cost of dependability or safety in a system, especially when measured against total life-cycle cost. Contributing to this also is the budgetary separation of procurement and post-deployment support in government systems.

The fault tolerance community lacks standards.

There is no standard symbology or a reference model, (e.g., the Open Systems Interconnect reference model developed for computer-based communications). There is no standard vocabulary to explain concepts such as fault tolerance, fault prevention, fault removal, fault avoidance, etc. There are efforts in this area (e.g. the glossary produced by IFIPS Working Group 10.4 on Dependable Computing and Fault Tolerance), but not general agreement. There are no general guidelines as to what can fail — including "real life" examples of how faults happen and advice to inexperienced engineers on what to protect a system against. There are no useful classifications of faults — application independent characterizations of faults. Standard fault tolerance architectures are not available to guide system designers. Life-cycle models do not typically include fault tolerance or the effect of dependable and safe operation. Handbooks of reusable components are not available.

Fault tolerant systems are still perceived as too complex.

Each scheme for fault tolerance is individually crafted, making it difficult to build on previous work. Although the concepts are common, the implementations are unique and system specific. Areas of complexity include: voting and synchronization, and N-version programming, which is popular but not well understood.

Fault tolerance looks deceptively simple.

This is a paradox. To some fault tolerance is too complex, to others it is too simple. Most system designers routinely incorporate some fault tolerance in their designs, but this experience does not necessarily equip them for designing ultra-reliable systems. Reliability engineers typically don't understand fault tolerant techniques.

3   State of the Practice

The main conclusic of the ᵣrkshop is that *the state of the practice is the state of the art,* that is, the practice is more *art* than *engineering.* Practitioners have been forced to use a variety of fault tolerance techniques without the assurance that the contribution made by these techniques to dependability or safety is worth the investment. Most fault tolerance approaches proposed by researchers have appeared in deployed systems. This includes techniques such as N-modular redundancy, recovery blocks, atomic transactions, N-version programming, self-checking hardware processors and self-checking process pairs. However, in some cases it appears that these techniques are not used widely enough, and in other cases it is not clear that the techniques have provided enough benefits to justify their cost.

The workshop reached a number of other significant conclusions. They are grouped below into four categories; application fault tolerance, hardware vs. software fault tolerance, deficiencies, and some general observations.

3.1   Application Fault Tolerance

- Successful examples of fault tolerant systems have existed for some time. These include the NASA Space Shuttle Data Processing System and the Mariner guidance and control system. Fragmented pockets of expertise exist.

- Generic solutions exist in certain domains such as space applications. Space vehicles such as Magellan have reused portions of prior designs (e.g. Mariner, NASA Space Shuttle Data Processing System design).

- Diverse design, both in hardware and software has been used in a number of deployed systems. It remains to be seen if its use is economically or technically justified. The benefits of physically separated resources are well accepted. The independence of logically independent resources is much more difficult to verify. One participant posed the question, "Can diverse hardware be justified given the cost of diagnostics for multiple versions of the hardware." Another participant reported that three versions of the system did not cost three times as much. There was at least a 30% reduction in cost because of commonality. In some cases people have used N-version programming approaches to reduce testing requirements (back-to-back testing).

- Fault tolerance in commercial aircraft systems has evolved from relatively simple approaches (monitored systems) to very sophisticated and complex approaches (diverse software). The designers have been driven to this extreme by the enormous economic penalty of a commercial aircraft loss traceable to its computer system.

- Military applications are less stringent than civilian aircraft applications. They generally rely on fault containment boundaries ("brick walling") for fault tolerance.

- Fault tolerance is beginning to move into application areas such as radar where mechanical failures have previously been the dominant concern.

- Fault tolerance is being considered for applications where data security is a major concern.

3.2 Hardware vs. Software Fault Tolerance

Hardware fault tolerance is mature, since it is relatively easy to insure that physical faults are independent. Software fault tolerance, which must deal with "generic" or common mode faults such as design faults, is not mature. Nevertheless, many hardware fault tolerance approaches require software for successful operation.

- The techniques for hardware fault tolerance are mature. When DRAMs were introduced into on-orbit applications, predictions of single bit upsets proved to be accurate and memory scrubbing techniques successfully compensated for them.

- The use of ultra-reliable components does not in itself provide dependability for ultra-dependable applications; fault tolerance mechanisms are also required. The Magellan spacecraft has sustained eight fault incidents involving four hardware failures and still continues to operate.

- Software fault tolerance appears to have emphasized two major options; 1) diversity and 2) recovery blocks. At least one of the workshop participants urged the development of fresh ideas, possibly derived from natural laws (e.g. physics).

3.3 Deficiencies

- There is no broadly accepted methodology for fault tolerant design or analysis. While there are concepts that are commonly used, there is no common design approach.

- Conventional design techniques do not take into account possible system failures. Successful fault tolerant design requires a constant awareness of what can go wrong throughout the design process. Failure domains are bigger than design domains. Designers must postulate failures, design, and iterate.

- Software reliability engineering is a new discipline, and most software reliability engineers don't understand fault tolerance.

- Verification is the weakest area. For example, there is no way to determine if the money spent for backup recovery software might be more effectively employed in improving the quality of the primary software. Fault tolerance cover-

age can be verified using either of two approaches; analytical or testing with simulation/fault injection. Complete formal proofs are seldom done; rather semi-formal proofs are used as gu. lines for reasoning. When they are used, it is important that they be carried throughout the life cycle to guarantee that future changes do not violate the assumptions.

- No matter how well we verify a system, users will still continue to insist on qualitative measures that provide a minimal level of independence (i.e. two independent success paths.) As one participant said, "If you provide 10^{-9} failures per year reliability in a system that is 'single string' (includes a single point of failure) it is very hard to convince somebody to trust their life to it."

- Requirements for fault tolerance appear in two forms; qualitative (i.e. fail soft, fail hard), or quantitative (i.e. seven nines probability of success). It is difficult to know when each is appropriate. Ideally, it must be possible to validate the requirements. It is impossible to prove quantitative requirements for the current state of the art.

3.4 General Observations

- Critical software is a multi-disciplinary, team effort. Fault tolerance is at the center of system design. Fault tolerant practitioners must also understand system hardware-software partitioning, system interfaces, and system performance requirements. They have to know how everything works normally, how the system is designed, and what can go wrong.

- Transients are usually more important to deal with than permanent failures. A Rand study reports that most problems in F-14, F-15, and F-16 avionics systems were "cannot duplicate" problems, presumably due to transients instead of permanent failures.

- There are many lists of fault tolerant mechanisms (e.g. the techniques directory in Volume III of the series *Dependability of Critical Computer Systems* published by Elsevier Science Publishers in 1990) which constitute a "toolkit" of techniques, but they are fragmented and applied on an ad hoc basis.

- Mature programs tend to add fault tolerance software to cope with scenarios that are more and more improbable and complex. For example, in a satellite application a ROM subsystem was provided as a recovery mechanism, despite the fact that no known scenario would lead to its use. In service the ROM has been required at least three times. This process of continuing refinement carries with it the risk of introducing bugs into previously working software.

- Fault tolerance can improve fault avoidance. The analysis of a system necessary for adding fault tolerance can often uncover flaws in the system's design.

- The system designer is the weak link in many industrial applications, since the designer often only implements a given design only once in a career. Large consulting firms often assign engineers with limited experience to critical

projects. Such a person is an expert in neither the application domain nor in fault tolerance. When the engineer moves on to the next project there is no opportunity to apply the lessons learned.

4 Fault Tolerance Technology Needs

4.1 Methodology

- Common reference material is needed. This should include: a standard vocabulary, a reference model, useful fault classes (i.e. an application-independent characterization of faults), guidelines as to what can fail (what to protect a system against), and examples of how faults happen in "real life".

- We need methods for effectively dividing systems into critical and non-critical subsystems. Without such a division, the high dependability system design and validation process must span an entire system, an unaffordable alternative. Furthermore, it is extremely difficult to extend and modify the noncritical portion of a system that lacks such a division.

- The design process must be extended to explicitly consider faults (what can go wrong) as well as functionality (what can go right).

- We have to recognize that fault tolerant systems will continue to be custom designs, but that they will be implemented using a few well-understood approaches. Methods of designing and analyzing standard fault tolerant subsystems are needed that can be reused often enough to justify extensive validation and testing. Standard subsystems offer the only practical way to accumulate sufficient service exposure to provide adequate confidence in a subsystem's reliability.

- Standard implementations of fault tolerant mechanisms, such as for synchronization and voting, are needed to deal with concerns that these approaches are too complex.

- We need methods for identifying fault containment regions.

- A validation process that supports the design process is needed. The effort expended in validating a design should not only provide the required confidence in the dependability of the design, but also inform designers of the strong and weak aspects of their design. Such a process will reinforce good design practices and discourage the use of potentially less fault tolerant approaches.

- The scope of formal proofs of correctness must be extended to cover a greater portion of system designs. As this is done, it will be essential to find ways of carrying the assumptions and properties used in the proofs through the life cycle to guarantee that upgrades do not violate critical assumptions.

4.2 Theory

- Diverse hardware and software have been incorporated into fielded systems for years, yet so far little evidence has been accumulated that suggest these particular measures are cost-effective. The value of diversity in general and diverse software in particular must be established.

- Fresh ideas (other than diversity and recovery blocks) must be developed for software fault tolerance. In particular we should seek an alternative to diverse software for generic or common mode faults.

- Fault tolerance needs to address real-time issues (i.e. fault latency) as well as functionality issues. Time to criticality is an essential concept.

- An economic model that recognizes not only the value and cost of functionality, but also the value and cost of dependability is needed. This will allow accountants and program managers to include fault tolerance intelligently in system procurements.

4.3 Education

- An easily accessed repository of past experience is needed to insure that past mistakes are not repeated and to provide reference material for new engineers entering the field. It is important that this collected experience be understandable. In other words, raw material is not sufficient.

- Fault tolerance must be added to computer science and engineering curricula. This must appear early enough in the curriculum to ensure that engineers learn to consider what can go wrong as well as what is to be accomplished in all phases of system design and implementation.

- A process is needed to educate users, especially the government, in the importance of fault tolerance. Fault tolerance should be recognized as a maturing technology that must be considered as an alternative or complement to fault avoidance and fault removal in achieving dependable systems. This process could include seminars, workshops, books, papers, videotapes, etc.

- The disciplines of reliability engineering and fault tolerance should share more concepts than they do at present, especially definitions of hazards, faults, system and software reliability, and failure models.

5 Potential SEI Role

Workshop participants were asked to suggest potential roles for the SEI in fostering the adoption of fault tolerant technology. The suggestions are grouped in the same categories as were used in the discussion of needs in Section 4

5.1 Methodology

- Develop a reference model for fault tolerant systems that allows us to partition a prototypical system into abstract functions that can individually be analyzed.

- Select and distribute a standard vocabulary and a standard fault classification scheme.

- Specify reusable fault tolerant services and foster their implementation in a demonstration environment.

- Define the role of fault tolerance in the overall system engineering process.

5.2 Theory

- Study the cost effectiveness of diversity in compensating for software faults.

- Assist in the development of accurate metrics of software dependability.

5.3 Education

- Help establish communications between practitioners, researchers, and program offices via a newsletter, workshops, etc.

- Serve as neutral party to collect, interpret, and distribute experience and lessons learned in the deployment of fault tolerant systems.

- Develop and distribute a fault tolerant curriculum module and associated educational material.

- Conduct an invitational seminar series on fault tolerance and distribute seminar tapes

6   Workshop Participants

Mr. Bob Dancey
IBM FSD
Building 869, Room 2F05
9221 Corporate Boulevard
Rockville, MD 20850
(301) 640-3944

Mr. Terry Devlin
Honeywell Air Transport Systems Division
21111 N. 19th Avenue
Phoenix, AZ 85027-2708
(602) 436-2496

Mr. George Gilley
The Aerospace Corporation
Mail Station MI-046
Post Office Box 92957
Los Angeles, CA 90009-2957
(213) 336-1552
FAX: 213-336-8266
e-mail: gilley @aerospace.aero.org

Mr. Jack Goldberg
3373 Cowper Street
Palo Alto, CA 94025
(415) 859-2784
e-mail: goldberg@csl.sri.com

Mr. Walter L. Heimerdinger
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213
(412) 268-6931
FAX: (412) 268-5758
e-mail: wlh@sei.cmu.edu

Mr. Larry James
Hughes Aircraft Corporation
Ground Systems Group - Bldg. 618, MS W308
P.O. Box 3310
Fullerton, CA 92634

Mr. Stephen B. Johnson
Martin Marietta Astronautics
MS: T200; P.O. Box 179
Denver, CO 80201
(303) 977-1449
FAX: 303-977-1530

Mr. Jim Kelley
Hughes Aircraft Corporation
Radar Systems Group
P.O. Box 92426
Los Angeles, CA 90009
(213) 334-2509

Mr. Ted Smith
IBM
Federal Sector Division, MC 6402A
3700 Bay Area Boulevard
Houston, TX 77058
(713) 282-8285

Mr. James E. Tomayko
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213
(412) 268-6806

Mr. Chris Walter
Allied- Signal Aerospace
9140 Annapolis Road
Columbia, MD 21045
(301) 964-4082

Mr. Charles B. Weinstock
Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213'
(412) 268-7719
FAX: (412) 268-5758
e-mail: weinstock@sei.cmu.edu

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified		1b. RESTRICTIVE MARKINGS None			
2a. SECURITY CLASSIFICATION AUTHORITY N/A		3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release Distribution Unlimited			
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A					
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CMU/SEI-91-SR-13		5. MONITORING ORGANIZATION REPORT NUMBER(S)			
6a. NAME OF PERFORMING ORGANIZATION Software Engineering Institute	6b. OFFICE SYMBOL (if applicable) SEI	7a. NAME OF MONITORING ORGANIZATION SEI Joint Program Office			
6c. ADDRESS (City, State and ZIP Code) Carnegie Mellon University Pittsburgh PA 15213		7b. ADDRESS (City, State and ZIP Code) ESD/AVS Hanscom Air Force Base, MA 01731			
8a. NAME OF FUNDING/SPONSORING ORGANIZATION SEI Joint Program Office	8b. OFFICE SYMBOL (if applicable) ESD/AVS	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F1962890C0003			
8c. ADDRESS (City, State and ZIP Code) Carnegie Mellon University Pittsburgh PA 15213		10. SOURCE OF FUNDING NOS.			
		PROGRAM ELEMENT NO 63756E	PROJECT NO. N/A	TASK NO N/A	WORK UNIT NO. N/A

11. TITLE (Include Security Classification)
Fault Tolerant Systems Practitioner's Workshop (June 10-11, 1991)

12. PERSONAL AUTHOR(S)
Heimerdinger, W.L., Weinstock, C.B.

13a. TYPE OF REPORT Final	13b. TIME COVERED FROM TO	14. DATE OF REPORT (Yr., Mo., Day) October 1991	15. PAGE COUNT 15

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse of necessary and identify by block number)
FIELD	GROUP	SUB. GR.	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

On June 10-11, 1991, a Fault Tolerant Systems Practitioner's Workshop was held at the Software Engineering Institute. The purpose of the workshop was to attempt to identify how fault tolerance is being applied today, why fault tolerance is under used, and what can be done to bring fault tolerant practices into wider use. Attendance at the workshop was limited to a small number of practitioners who had successfully applied fault tolerance in a systems context. This report summarizes the proceedings of the workshop which included a discussion of barriers to the deployment of fault tolerant systems, a summary of the state of the practice, and a discussion of the technology needs of fault

(please turn over)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ■ SAME AS RPT □ DTIC USERS ■	21. ABSTRACT SECURITY CLASSIFICATION Unclassified, Unlimited Distribution		
22a. NAME OF RESPONSIBLE INDIVIDUAL Charles J. Ryan, Major, USAF	22b. TELEPHONE NUMBER (Include Area Code) (412) 268-7631	22c. OFFICE SYMBOL ESD/AVS (SEI)	

ABSTRACT —continued from page one, block 19

tolerance. The report concludes with a discussion of ways the Software Engineering Institute may be able to help bring fault tolerant practices into wider use.

ABSTRACT —continued from page one, block 19

tolerance. The report concludes with a discussion of ways the Software Engineering Institute may be able to help bring fault tolerant practices into wider use.